NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

A COMPARATIVE ANALYSIS OF THE SNORT AND SURICATA INTRUSION-DETECTION SYSTEMS

by

Eugene Albin

September 2011

Thesis Advisor:	Neil Rowe
Second Reader:	Rex Buddenberg

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2011	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE A Comparative Analysis of the Snort and Suricata Intrusion-Detection Systems		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Eugene Albin			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number NA.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE** A	

13. ABSTRACT (maximum 200 words)

Our research focuses on comparing the performance of two open-source intrusion-detection systems, Snort and Suricata, for detecting malicious activity on computer networks. Snort, the de-facto industry standard open-source solution, is a mature product that has been available for over a decade. Suricata, released two years ago, offers a new approach to signature-based intrusion detection and takes advantage of current technology such as process multi-threading to improve processing speed. We ran each product on a multi-core computer and evaluated several hours of network traffic on the NPS backbone. We evaluated the speed, memory requirements, and accuracy of the detection engines in a variety of experiments. We conclude that Suricata will be able to handle larger volumes of traffic than Snort with similar accuracy, and thus recommend it for future needs at NPS since the Snort installation is approaching its bandwidth limits.

14. SUBJECT TERMS Intrusion-detection System (IDS), Snort, Suricata, Information Technology, Information Assurance, Network-Security Monitoring (NSM), Intrusion Prevention System (IPS)			**15. NUMBER OF PAGES** 68
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

A COMPARATIVE ANALYSIS OF THE SNORT AND SURICATA INTRUSION-DETECTION SYSTEMS

Eugene Albin
Lieutenant Commander, United States Navy
B.M., Southwestern University, 1995

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
September 2011

Author: Eugene Albin

Approved by: Neil Rowe, PhD
Thesis Advisor

Rex Buddenberg
Second Reader

Dan Boger, PhD
Chair, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Our research focuses on comparing the performance of two open-source intrusion-detection systems, Snort and Suricata, for detecting malicious activity on computer networks. Snort, the de-facto industry standard open-source solution, is a mature product that has been available for over a decade. Suricata, released two years ago, offers a new approach to signature-based intrusion detection and takes advantage of current technology such as process multi-threading to improve processing speed. We ran each product on a multi-core computer and evaluated several hours of network traffic on the NPS backbone. We evaluated the speed, memory requirements, and accuracy of the detection engines in a variety of experiments. We conclude that Suricata will be able to handle larger volumes of traffic than Snort with similar accuracy, and thus recommend it for future needs at NPS since the Snort installation is approaching its bandwidth limits.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION....1

II. PREVIOUS WORK....7

III. DESCRIPTION OF METHODOLOGY AND EXPERIMENTS....13
- A. EXPERIMENTAL SETUP....14
- B. EXPERIMENTS....16

IV. DISCUSSION OF RESULTS....21
- A. EXPERIMENT ONE....22
- B. EXPERIMENT TWO....27
- C. EXPERIMENT THREE....30

V. CONCLUSION....37
- A. DISCUSSION....37
- B. RECOMMENDATION....38
- C. FUTURE RESEARCH....39

APPENDIX A....41
- ANNEX 1 TABLE OF COMBINED SURICATA AND SNORT ALERTS DURING TESTING....41
- ANNEX 2 EXAMPLE PYTBULL REPORTS FOR SURICATA AND SNORT....42

APPENDIX B....45
- ANNEX 1 EMERGING THREATS (ET) AND VULNERABILITY RESEARCH TEAM (VRT) RULE CATEGORIES USED IN EXPERIMENTS.....45

LIST OF REFERENCES....47

INITIAL DISTRIBUTION LIST....49

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1. Suricata multi-thread design (From OISF, 2011c) 9
Figure 2. Suricata multi-CPU affinity (From OISF, 2011c) 9
Figure 3. Experiment One setup....17
Figure 4. Experiment Three logical network diagram 18
Figure 5. Suricata CPU Use 22
Figure 6. Snort CPU Use....23
Figure 7. Suricata RAM Use 23
Figure 8. Snort RAM Use....24
Figure 9. Suricata Packet Rate 25
Figure 10. Snort Packet Rate....25
Figure 11. Suricata and Snort Combined Alert Frequency 26
Figure 12. Suricata runmode performance for 48 CPUs....28
Figure 13. Suricata runmode performance for 4 CPUs....29
Figure 14. Suricata detect thread ratio performance 4-CPU 29

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1. Summary of Snort Alerts 31
Table 2. Summary of Suricata Alerts 31
Table 3. Snort and Suricata Recall and Precision 32

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

CPU	Central Processing Unit
CVE	Common Vulnerabilities and Exposures
DHS	Department of Homeland Security
DoS	Denial of Service
ERN	Education Research Network
ET	Emerging Threats
FTP	File Transfer Protocol
HIDS	Host-based Intrusion-Detection System
HTTP	Hyper Text-Transfer Protocol
ICMP	Internet Control Message Protocol
IDS	Intrusion-Detection System
IP	Internet Protocol
IPS	Intrusion-Prevention System
LFI	Local File Inclusion
MSSQL	MicroSoft Structured Query Language
NSM	Network-Security Monitoring
NIC	Network Interface Card
NIDS	Network-based Intrusion-Detection System
NPS	Naval Postgraduate School
OISF	Open Information Security Foundation
PCAP	Packet CAPture
PCRE	PERL Compatible Regular Expression
PDF	Portable Document Format

RAM	Random Access Memory
RPM	Resource Package Manager
SID	Snort IDentification
SPAWAR	Space and Naval Warfare Command
SQL	Structured Query Language
SSH	Secure Shell
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
VM	Virtual Machine
VNC	Virtual Network Computing
VRT	Vulnerability Research Team
VTY	Virtual Teletype Terminal
YAML	YAML Ain’t Markup Language

ACKNOWLEDGMENTS

This project would not have been possible without the guidance and support of my thesis advisors, Dr. Neil Rowe and Mr. Rex Buddenberg, without whom this paper would have been a jumbled mess of ideas. I'd also like to thank Chris Gaucher, Director of Cyber-Security and Privacy at NPS, and his team including Jason Cullum, Simon McLaren, for granting me permission to conduct this research on the NPS network and secondly for all of the ideas and discussion about intrusion detection here at NPS. Thanks also to Lonna Sherwin and her server management team including Eldor Magat for providing access to the hardware needed to conduct these experiments. Thanks for answering all of my questions about server virtualization and configuration. I'd also like to thank the many participants in the OISF user's mailing group, especially Peter Manev, Will Metcalf, Dave Remien, and Victor Julien for putting up with my endless barrage of questions and providing me with the help that I needed to understand the nuances of Suricata.

Most importantly I want to thank my amazing wife, Elizabeth, for her endless patience and understanding as I spent countless days and nights away from our family. Your love and support gave me the strength to make it to the end. Finally, thanks go to my two boys, Elliott and Sean, the pride of my life, who have selflessly given up their summer with Papa so I could complete my thesis. I love you both.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

In spite of the many developments in network security over the past decade, the Internet remains a hostile environment for our networked computer systems. According to the Symantec Internet Security Threat Report for 2010, "The volume and sophistication of malicious activity increased significantly in 2010" (Fossl, 2011). Attacks by worms such as Stuxnet and exploits in commonly used programs such as Adobe Acrobat are recent examples of the caliber and frequency of malicious activity that is seen being crafted with today's technology. It is no longer safe to ignore the security threats that we face, and we continue to become more and more dependent upon network connectivity and the Internet.

The threats are not only to computers and hardware that we connect to the Internet, but to the data and information that resides within that infrastructure. More and more we as a society are growing and developing our digital presence. Beyond just our email, shopping habits, and bank account information, the data that is collected about us, that defines us, exists in this hostile network of systems; and without a commensurate increase in technologies to protect that data we risk compromise, theft, exploitation, and abuse of the data that defines our digital selves, be it our individual, personal identity or our corporate digital self, will cause real and significant damage in the real world.

The Naval Postgraduate School is one such institution where information is processed on a large scale, stored, and transmitted over a diverse computer network. Information security is crucial to protect and sustain the development of critical research. Like many other government organizations the Naval Postgraduate School is constantly being probed and attacked in an attempt to penetrate the defenses and obtain the information within the NPS information domain. To defend against that, NPS has built a robust defense architecture that monitors and guards the critical information against these intrusion attempts. However this threat is not stagnant, and will continue to grow, change, and adapt to the current network security technologies.

Consequently, we must continue to advance the development of new security technologies to defend against the rising tide of malicious activity penetrating those networks. Best practices in network security dictate that "defense-in-depth" (the strategy of establishing multiple layers of defense around critical infrastructure to protect the data) is an effective posture in defending against these attacks. One critical aspect of network-security monitoring is the incorporation of intrusion-detection and intrusion-prevention technologies within our defense-in-depth strategy (Kuipers & Fabro, 2006; Kumar & Panda, 2011). An intrusion-detection system (IDS) monitors and logs the traffic that is traversing a network for signs of malicious or unwanted activity, and generates an alert upon discovery of a suspicious event.

There are two types of intrusion-detection systems, host-based and network-based. A host based intrusion-detection system is a tool that resides on a network node, or a computer that is connected to the network. Similar to a virus or malware scanner, it scans traffic destined for that particular host for signs of malicious activity, then generates alarms for those events. At an enterprise scale, these host-based systems are widely deployed to send reports back to a centralized monitoring node where aggregation and study of the collective threat picture can occur. A networked-based intrusion-detection system is a device connected to the network in a manner similar to a network-protocol analyzer, or "sniffer" as it is commonly called. But it goes one step beyond simple packet capture and presentation to examine the contents of the packet data for signs of malicious activity. A network-based intrusion-detection system monitors all of the network traffic and upon sensing an intrusion, sends an alarm to a monitoring console for further action. Multiple network-based intrusion detection systems can be deployed throughout an enterprise at critical network junctures: the boundary link(s) to the Internet, the trunk to the VIP computer systems, the ingress and egress points for the server farm or data center, or the demarcation point of the enterprise wireless infrastructure.

Intrusion detection as part of network-security monitoring involves reviewing and examining large amounts network traffic data. There are a number of ways to do network-security monitoring using intrusion-detection engines. One is to monitor network traffic in real time using a variety of tools that examine and interpret the traffic,

and output alerts as malicious traffic crosses the sensor on the network. This real-time monitoring allows for an immediate response to any alarms generated by the intrusion detection engine. The processing speed for real-time monitoring is bounded by the maximum speed of the network interface card. Should the engine reside on a system with a low-capacity network interface card, the card may quickly become overloaded with network traffic and begin to drop packets. The more packets that are dropped, the greater the chances of a malicious payload getting through the intrusion-detection layer of defense. Therefore, the system running the detection engine should be capable of processing traffic at a speed equal or greater to the maximum capacity of the network.

Another method is to use the intrusion-detection system engine for playback or non-real-time analysis of archived traffic. This approach is more often applied to post-incident network forensic analysis. Where real-time monitoring of network traffic is bounded by the speed of the network interface card, forensic analysis of archived network traffic is limited by the computing hardware and the detection engine software. It is during forensic analysis that we will see the greatest performance increase by increasing the computer's processing ability in CPU, memory, and disk I/O speeds

There are presently two main categories of intrusion-detection, anomaly-based and signature-based. Anomaly-based detection examines the network traffic from a holistic perspective, looking for traffic that falls outside of what is considered normal activity. Any such events are analyzed and if necessary, further investigation and subsequent action is taken to mitigate the anomaly (García-Teodoro, Díaz-Verdejo, Maciá-Fernández, & Vázquez, 2009). Anomaly detection is good for discovering new, previously unknown attacks in a relatively small network environment. Signature-based intrusion-detection attempts to match network traffic data to a preloaded signature database. Typically, these signature rules are generated from previously discovered malicious traffic, but they can be custom crafted to match any sort of traffic flowing through the network. Upon matching a signature rule the detection engine generates an alert which is subsequently sent to the analyst for further action.

An example of a signature-based detection would be when the intrusion-detection system generates an alert from an attempt by an attacker to create a reverse command

shell to an internal server. In this case, the attacker is attempting to gain unauthorized access to a protected system within the network perimeter by attempting to pass to the server a series of commands which would initiate a reverse connection to the attacker's computer. This type of attack could be detected by a signature-based system through the commands that the attacker issues. One common command that a system scans for is "c:/windows/cmd.exe." An anomaly-based system would instead notice a command shell request being generated from outside of the protected server enclave. Since this is not considered normal activity for the traffic that flows to and from the Internet, this would then be flagged as an alert. Other examples of behavior that would trigger an anomaly-based alert might include after-hours access by a user who normally works during the day, indicating possible unauthorized access; significant change in data traffic from one area of the network to another, indicating possible data exfiltration; or an increase in data communications between a growing number of workstations, indicating a possible worm infestation.

In recent years, with the increase in complexity and frequency of Internet attacks, intrusion detection has become significantly more important to a wider audience. Numerous companies and organizations have been working to develop the technology and have produced several products, both open source and proprietary. One of the most popular and widespread open-source signature-based network intrusion-detection engines is Snort, maintained by SourcFire (www.sourcefire.com). Originally developed to monitor the application layer of network data packets, Snort was developed in 1998 by Martin Roesch and is based on the Libpcap library (Roesch, 2005). The current modular design of Snort in today's version was settled on in 1999 with Snort 1.5. This modular design allows developers to build and add-on additional features without the need to rewrite the core detection engine. Snort has become the de-facto industry standard for signature-based network intrusion-detection engines. An overview of the specific capabilities of the Snort intrusion-detection system is in (Tenhunen, 2008).

Almost a decade later, in 2009, the Open Information Security Foundation (OISF) released a new signature-based network intrusion-detection engine called Suricata. Suricata is also an open-source signature-based network intrusion-detection engine

envisioned to be the next generation intrusion-detection/prevention system engine (Jonkman, 2009). Significant funding for the project comes from the Department of Homeland Security (DHS) Directorate for Science and Technology (S&T) Homeland Open Security Technology (HOST) program and the Navy's Space and Naval Warfare Command (SPAWAR). As the OISF title implies, the development framework for Suricata is open-source and is licensed by the GNU Public License v2 (OISF, 2011a). One advance that Suricata incorporates is a new Hyper-Text Transfer Protocol (HTTP) normalizer. Called the HTP library and developed independently for the Suricata project by Ivan Ristic, it is an advanced HTTP parser developed for Suricata and the OISF that is designed to be "security-aware," meaning that it is capable of examining HTTP traffic for the attack strategies and evasion techniques used by attackers to circumvent an intrusion-detection system (Ristic, 2009).

Another advance in the Suricata engine is the ability to employ native multi-threaded operations, something more necessary as network bandwidth increases (Nielsen, 2010). The typical Snort installation can process network traffic at a rate of 100-200 megabits per second before reaching the processing limit of a single CPU and dropping packets to compensate (Lococo, 2011). That is because the current Snort engine is a single-threaded multi-stage design (Roesch, 2010) and does not perform as well as Suricata in a multi-threaded environment (Day & Burns, 2011). For Snort to take advantage of the multiple processors, one would have to start a new instance of Snort for each desired CPU, which could be a management challenge. Suricata is designed from the outset to take advantage of operating with multiple CPUs (OISF, 2011a). This required development of original detection algorithms from the ground up. Nonetheless, the developers intend to support the same rule language used in the Snort rules; and when the Suricata engine is more stable the OISF will make available Suricata's extended features (OISF, 2011a).

In Chapter II, we will review the challenges of intrusion-detection and look at how Suricata and Snort attempt to address these challenges. We will examine other works in the field of network security that compare the differences between Suricata and Snort.

In Chapter III, we will introduce and describe our testing methodology for our comparison of Suricata and Snort. We will discuss the setup of our experiments and the steps involved in building the testbed. We will also introduce the supporting applications required to complete our experiments.

In Chapters IV and V, we will present our results and conclusions respectively. We will then discuss further research that can be done to compare the two engines, and provide a recommendation for implementation of the Suricata intrusion-detection system.

II. PREVIOUS WORK

Intrusion detection is difficult to accomplish perfectly. With the volume of network traffic rapidly increasing and the number and complexity of network attacks increasing just as quickly, it becomes increasingly difficult for a signature-based intrusion-detection system to keep up with the current threats (Weber, 2001). When a system fails to generate an alarm, the result could be the compromise of critical data within the network infrastructure. Worse, if the system generates too many false alarms, the operators monitoring the system may become desensitized to the alerts and may ignore a genuine alert in the future.

Four possible situations occur with alerts generated by an intrusion-detection system: true-positive, true-negative, false-positive, and false-negative. The true positive is when the detection engine generates an alert based on the correct identification of a potential threat. The true negative is when a detection engine does not generate an alert for normal traffic: this occurs during a benign network traffic flow. The false positive is when a detection engine generates an alert for an event that is not malicious. The most dangerous of conditions is the false negative when a detection engine does not alert on malicious traffic, thereby allowing it to enter the network without notice. Accuracy in intrusion detection can be measured by the number of false positives and false negatives generated by the detection engine.

There are a number of ways that false positive and false negative conditions can occur. Faulty rule design, including invalid signature information or improper rule language, is one way, but this is rare. Similarly, rules may be imprecise because there is no distinctive signature for a particular kind of attack. These types of error would result in a false positive or false negative across all intrusion-detection engines using the same rules. False positives and false negatives can also occur through performance problems with the detection engine itself. To analyze a network traffic flow for signs of intrusion, the detection engine must examine every packet that traverses the network wire. If the hardware fails or becomes otherwise overused, it can drop one or more packets, allowing a malicious packet to enter the network. Unlike a dropped packet in a router, in an

intrusion-detection system a dropped packet is not discarded at the detection engine, but rather when an intrusion-detection engine drops a packet it passes through to the protected network. This overload condition could be intentionally caused by an attacker for the purpose of injecting malicious traffic. It is therefore important to measure the performance capability of an intrusion-detection system to ensure that the system can support the capacity of the network on which it is deployed.

One means to improve performance is to split the data stream between multiple processing engines. However, any malicious bits that may be residing in a traffic flow may get split too, resulting in neither engine matching the complete signature. To mitigate this situation, either the detection engines must pass coordination information between each other, further increasing the computational load on the system, or the network traffic must be divided by a "flow-aware" network tap that can keep related data together.

This research examines the Suricata intrusion-detection system (Shimel, 2010). By containing the multiple threads within the same detection engine, a multi-threaded detection engine can make intelligent decisions on how to split processing and can coordinate signature detection between these threads all within the same detection engine. Moore's Law (Moore, 1965) predicts that computational speed doubles every eighteen months for single threaded processing architectures. Multi-threaded processing can take advantage of that prediction. According to Nielsen's Law of Internet Bandwidth, we will also see 50% increases in network bandwidth every year (Nielsen, 2010). The performance of our intrusion-detection system systems should increase as our demand for network bandwidth also increases. The developers of Suricata have chosen to address that demand through multi-threaded processing (OISF, 2011a).

Considering that the most computationally intensive work performed by an intrusion-detection engine is detection, the Suricata developers decided to use threads for detection. Figure 1 gives an example with the creation of three detect threads. Suricata can receive network traffic from the network interface card or from previously recorded network traffic from a file stored in PCAP format. Traffic is passed through the decode

module where it is first decoded as per its protocol, then the streams are reassembled prior to being distributed between the signature-detection modules.

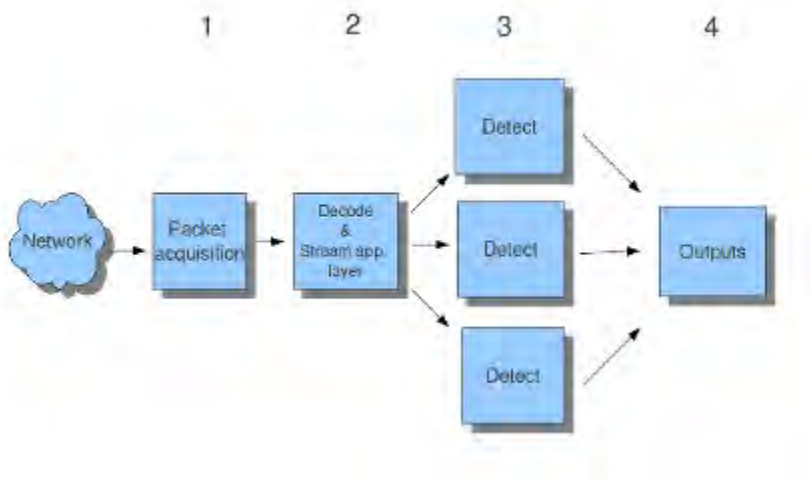

Figure 1. Suricata multi-thread design (From OISF, 2011c)

The Suricata configuration file allows the user to configure which and how many threads, and how many CPUs will be involved in the processing of each stream. Figure 2 illustrates how Suricata can distribute the various modules in the processing stream across the different CPU cores in the computer.

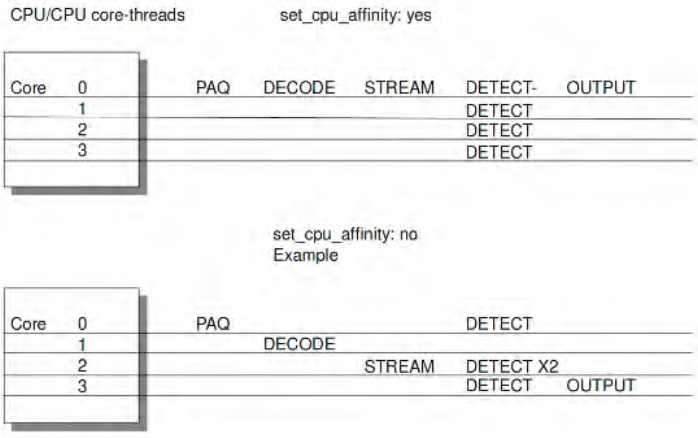

Figure 2. Suricata multi-CPU affinity (From OISF, 2011c)

Previous work compared the performance of Snort version 2.8.5.2 to Suricata version 1.0.2 (Day & Burns, 2011). Their testbed consisted of an Ubuntu 10.04 virtual machine hosted in a VMWare Workstation 6.5 virtual environment running on a 2.8GHz Quad-Core Intel Xeon CPU with 3GB of RAM. The study examined detection engine speed as well as the accuracy under varying degrees of network and processor use. CPU use was controlled using Cpulimit (cpulimit.sourceforge.net), network bandwidth was controlled using Tcpreplay (tcpreplay.synfin.net), and alert generation was stimulated by injecting six known malicious exploits generated using the Metasploit framework (www.metasploit.com). The results show that Snort is more efficient with system resources than Suricata, and when operating in a multi-CPU environment, Suricata is more accurate due to fewer false negative alerts. However they concluded that the overall performance of Suricata in a four-core environment was slower than that of Snort in a single-core environment when processing 2 gigabytes of previously captured network data.

In April 2011, Damaye (2011b) published an online report comparing the accuracy of Snort and Suricata in detecting a wide variety of malicious files and suspicious actions. His tests incorporated the latest versions of Suricata and Snort with signature rules from both the Snort Vulnerability Research Team (VRT) and Emerging Threats (ET). Creating a custom application in Python specifically designed to send a variety of specific vulnerabilities through an intrusion-detection system via a number of different vectors, he attempted to measure the accuracy of both detection engines. During his tests he measured the number of true and false positives and negatives and assigned a score to Snort and Suricata for each of the tests conducted. The results of his study concluded that the two rule sets (VRT and ET) worked well together but required tuning to be most effective. He further concluded that while Suricata is a promising new technology with key features, Snort still is preferable for production environments.

Leblond (2001) ran a series of tests to examine the performance of the multi-threading capabilities of Suricata. By adjusting the detect_thread_ratio and the cpu_affinity variables in the Suricata configuration file on a dual 6-core CPU system with hyper-threading enabled, he was able to achieve the best performance by reducing the

thread ratio to .125, which corresponded to three threads generated by Suricata in his test environment. He also determined that the hyper-threading configuration caused variations in the performance results (30% variation between runs) and that it was actually best to configure the multiple threads to run on the same hardware CPU. Following his initial research, in February 2011 Leblond dove deeper into the workings of the Suricata multi-threading design. Using a tool called Perf-tool (code.google.com/p/google-perftools) he was able to determine that, as the number of threads increased, more time was spent waiting for an available lock. His conclusion was that configuring Suricata to run in RunModeFilePcapAutoFp results in a steady performance increase, whereas RunModeFilePcapAuto shows an initial increase, then a continued decrease in performance as measured by packets per second processed.

THIS PAGE INTENTIONALLY LEFT BLANK

III. DESCRIPTION OF METHODOLOGY AND EXPERIMENTS

Our experiments tested and compared the Suricata and Snort intrusion-detection engines in performance and accuracy in a busy virtual environment. The experiments evaluated performance by measuring the percentage of CPU use, memory use, and network use. We measured accuracy by subjecting both detection engines to malicious traffic in controlled tests, and comparing the alerts generated by each application.

Traffic data for our experiments originated from a network tap located on the backbone of the Naval Postgraduate School (NPS) Education Resource Network (ERN). At the point of the network tap the NPS ERN has a bandwidth of 20 Gbps, providing a large "pipe" to send traffic to our intrusion-detection system. Traffic across the NPS backbone averages 200Mbps per day. We used this traffic to compare the performance and alerts of the virtual machine while running Snort and Suricata.

Signatures for the experiments came from the two primary open-source intrusion-detection system rule maintainers, the SourceFire Vulnerability Research Team (VRT) and Emerging Threats (ET). The SourceFire VRT "develops and maintains the official rule set" for Snort (SourceFire, 2011). Emerging Threats originally began as a community-authored list of rules to augment the SourceFire VRT body of rules. Initially considered less robust than the VRT rules, the ET rules now provide new capabilities. Where the VRT rule set is specifically designed to support Snort exclusively, the ET rule set is "platform agnostic" by design and will work on any type of open-source intrusion-detection system application (Emerging Threats, 2011). Both Suricata and Snort support the rules from VRT and ET.

Our research questions are:

1. Does Suricata with its multi-threaded processing perform better than Snort with its single-threaded processing?

2. As CPU and memory use increase, are there differences in the number of dropped packets between the two engines?

3. Will Suricata handle heavy loads better than Snort?

4. Is Suricata suitable to be implemented within the NPS production network? At the time of the experiment, the NPS Information Technology Department was unsure whether to use Suricata and was hoping our experiments would help them to decide.

A. EXPERIMENTAL SETUP

Most experiments were conducted in a virtual machine running VMware ESXi 4.1. The server hardware was a Dell Poweredge R710 dual quad-core server with 96 GB of RAM. Each CPU was an Intel Xenon E5630 running at 2.4 Ghz. The data storage was accomplished through three fiber-channel attached RAID 5 configured arrays, supporting the relatively large network traffic capture files (PCAP files) needed to test the detection engines. The server had eight 1Gbps network cards installed, with four reserved for the various management activities and four available for the Virtual Machine. For our experiments the server was configured with two interface cards, one for system administration and one to capture network traffic. The interface card attached to the network tap was configured in promiscuous mode to allow it to receive all of the network traffic. Our virtual machine used 4 CPU cores and 16GB of RAM. The operating system chosen for the experiment was CentOS 5.6 due to its popularity for enterprise applications and close relationship to Red Hat Enterprise Linux.

Installation of Suricata was relatively straightforward. Precompiled versions of Suricata were difficult to find due in part to the relative newness of Suricata compared to Snort. So we compiled it. To do this we had to install a number of software dependencies which were not included in the CentOS 5.6 distribution. Among these were the Perl compatible regular expression (PCRE) libraries, packet-capture libraries (Libpcap) to allow the operating system to capture all of the traffic on the network, and YAML Libraries (libyaml) required to interpret Suricata's YAML-based configuration files (Ben-Kiki, Evans, & dot Net, 2010).

Initially, we started with version 1.0.3 of Suricata, but during the experiment the Suricata developers released a major version change to 1.1 beta2. This version fixed several key performance and rule issues that were present in the earlier versions (OISF,

2011a). After the initial installation the upgrade process was simple, involving only a download of the 1.1beta2 source code and compiling it on the system.

Installation of Snort 2.9.0.5 on CentOS Linux distributions has a number of known issues, namely compatibility with the version of Libpcap that is distributed with CentOS prior to 5.6. Fortunately, Vincent Cojot maintains a series of RPMs (precompiled software for installation on Red Hat-based Linux distributions) for Snort on CentOS (vscojot.free.fr/dist/snort). Installation of Snort simply involved downloading the latest Snort RPM and extracting the program. The required PCAP library was already installed during the Suricata install so there were no other dependencies involved. During the experiments the need to upgrade Snort 2.9.1 beta also became apparent due to the large size of our PCAP files. Unfortunately Vincent Cojot's RPM repository did not contain the latest 2.9.1 beta version of Snort at the time, so we were unable to upgrade to 2.9.1 and conduct some of our planned tests.

Care must be taken to ensure that the proper version of rules is downloaded for the corresponding intrusion-detection engine, or a significant number of errors will be reported during startup in Suricata, and in the case of Snort the startup process will abort altogether. The VRT web site does not maintain a separate set of rules optimized for Suricata, so upon loading the VRT rules in Suricata we received a number of rule errors.

We also used Pytbull, a utility written in Python and designed by Sebastian Damay to test and evaluate an intrusion-detection system's ability to detect malicious traffic (Damaye, 2011a) by sending it sample traffic. Installation of Pytbull was fairly simple considering that several of the dependencies for Pytbull were already installed for Suricata and Snort. Pytbull requires Python and Scapy (www.secdev.org/projects/scapy), and the environment must have an FTP server and a web server available.

To capture live traffic from the network and replay that data for static file analysis, the Tcpdump (tcpdump.org) and Tcpreplay (tcpreplay.synfin.net) utilities were also required. Collection of the system performance data while running each of the experiments was accomplished using the tool Collectl (Collectl.sourceforge.net).

B. EXPERIMENTS

The first experiment examined the real-time performance of each system independently while monitoring live backbone traffic from the NPS ERN. Performance data from the CPU, RAM, and network interface was recorded, examined, and compared. A variation of the first experiment ran both detection engines simultaneously.

The second experiment ran Suricata on the NPS Hamming supercomputer. The NPS High Performance Computing Center operates a Sun Microsystems 6048 "blade" system with 144 blades and 1152 CPU cores (Haferman, 2011) running CentOS 5.4 as the operating system. For our experiment we used one compute node composed of 48 AMD Opteron 6174 12-core processors with 125GB of RAM available. We measured the increased performance when running Suricata on this high-performance computer. The goal was to determine if it was feasible for an intrusion analyst to process stored network traffic significantly more quickly in such an environment. This task is important for our Information Technology Department as they are regularly called upon to do retrospective analysis of data of particular attacks.

The third experiment measured how well each intrusion-detection system detected a variety of malicious packets sent to it. This experiment was not concerned as much with the computational performance as with the accuracy of detection.

Experiment One Setup

The first experiment compared Suricata to Snort when monitoring network traffic at the NPS border router. The NPS backbone connects to the Internet with a maximum bandwidth of 20Gbps. Figure 3 shows the logical network diagram for Experiment One.

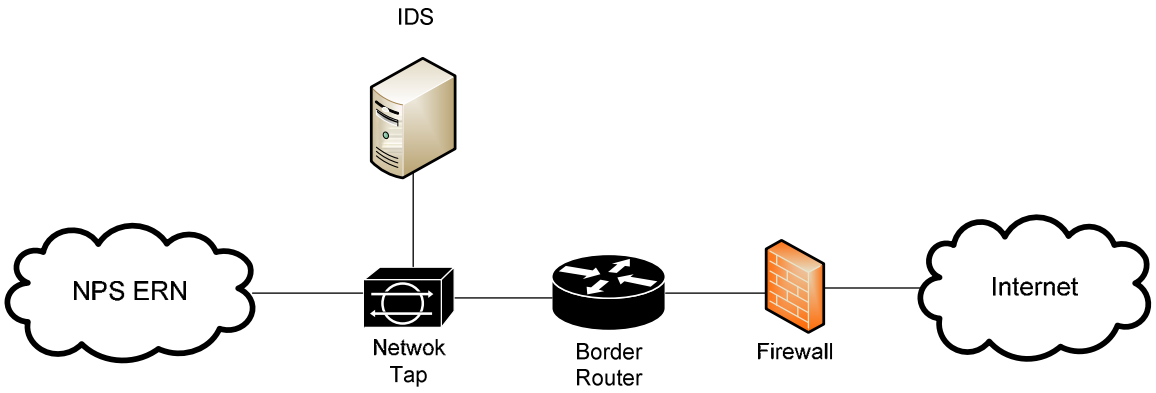

Figure 3. Experiment One setup

We first installed and ran each detection engine individually in the virtual machine environment with the combined ET and VRT rule sets. Collectl was used to record the CPU, RAM, and network use of the server. The experiments were conducted over approximately a 4 hour period of normal network use on the NPS backbone. Performance configuration settings for each detection engine were set to the default parameters.

We then ran instances of both Snort and Suricata on the virtual machine at the same time. This allowed us to compare the accuracy of each detection engine in generating alerts from the same live network traffic. System CPU, RAM and network use were also recorded for this experiment, but are not a reliable indicator of the true intrusion-detection system load since it is unusual to run two engines on the same system at the same time. We then evaluated the alert logs from each detection engine looking for differences.

Experiment Two Setup

For Experiment Two, we put Suricata on the Hamming supercomputer to measure the speed of processing there. We used Tcpdump to capture a large PCAP file from the NPS ERN backbone. The file was roughly 6GB and consisted of full packet data (obtained by `tcpdump -nnvi eth0 -s0`), the same type of data stored in a typical network archive. To reduce the impact of disk input/output latency we copied the PCAP file to a RAM disk on the Hamming computer. We ran Suricata with this large PCAP file

on both our Experiment One setup and on the Hamming supercomputer, and compared the time it took to analyze the file using the full rule set with various configuration settings.

Experiment Three Setup

The third experiment tested how accurately Suricata and Snort recognized malicious or irregular traffic. Using Pytbull we generated a number of tests containing suspicious or malicious payloads, and sent them through the intrusion-detection systems to stimulate alerts. These tests were divided into nine categories: client-side attacks, common rule testing, malformed traffic, packet fragmentation, failed authentication, intrusion-detection system evasion, shell code, denial of service, and malware identification. Each category contains a number of different tests for evaluating our detection engines.

Setup for this experiment required two additional machines: one to generate the test traffic (Client), and one to host an HTTP server with malicious PDF files (Hostile Internet web server) as illustrated in figure 4. We used a VMWare Workstation7 virtual machine running Ubuntu 10.04 for the client machine, and for the web server with the PDF files we used a Dell Latitude laptop running Xubuntu. This test required an FTP service and a web server be installed and running on the intrusion-detection system server. We chose to install Vsftpd for our FTP client due to its small size and ease in configuration. Fortunately CentOS 5.6 already had a web server included in the base distribution. To log the computational performance data we ran Collectl.

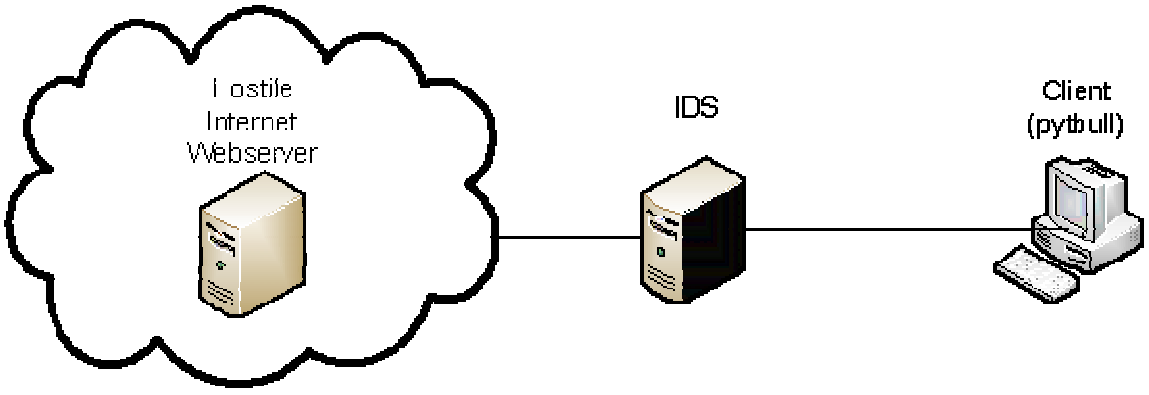

Figure 4. Experiment Three logical network diagram

For the experiment, our web server was preloaded with a variety of corrupt files. These files consisted of segments of observed malware from security-related sites on the Internet that collect these files for research purposes. Each tainted file was hashed prior to distribution to ensure the integrity of the file. For our experiment we selected some typical file types seen on the Internet, specifically four PDF and one XLS file.

Running the experiment consisted of starting Collectl on our system testbed computer to log the performance data, then starting the Pytbull client-side remote shell script there. Next we started the Httpd service on both the hostile web server and the intrusion-detection server. In addition, we start the Vsftpd service on the intrusion-detection server. After confirming that these services were running, we started our detection engine, either Snort or Suricata. Once the detection engine was loaded and listening to the network interface, we ran Pytbull from out testing client, pointing it at the address of our intrusion-detection system. The application completed the battery of tests, exited, and generated an HTML report listing the exploits that were attempted and the alert response from the intrusion-detection system if any. Then we stopped the HTTP services and the intrusion-detection services.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DISCUSSION OF RESULTS

Our experiments occurred over several days. As the experiments progressed we ran into a few issues while tuning Suricata in the virtual environment. We encountered an apparent upper limit in the amount of RAM that applications can use in a 32-bit operating system. As a result we were unable to load the entire combined ET and VRT rule set (more than 30,000 rules) in our 32-bit CentOS 5.6 operating system. The limitation is due to a 4GB memory limit for running both applications and the kernel in 32-bit Linux operating systems. (Suricata if compiled on a 64-bit operating system could take advantage of up to 48GB of RAM and could accommodate over 30,000 rules.) As a result, for our experiments we had to reduce the number of rules to a combination of ET and VRT rules totaling 16,996 signatures. The list of rule files used in our experiments can be found in Appendix B Annex 1.

We also ran into a problem with keeping up with network traffic, which limited our ability to effectively measure the accuracy of both the Snort and Suricata detection engines. While in promiscuous mode, the kernel was unable to buffer the entire network stream as it was passed from the network interface card. Using Tcpdump, we first saw dropped packets at rates of up to 50%, and both Suricata and Snort indicated high numbers of dropped packets in their log files. Considering the commonality of the high rate of packet drops, we concluded that the cause was in the virtual networking environment of the ESXi server. Attempting to mitigate this problem, we adjusted several kernel settings on the server to increase the memory allocated to the networking buffer. The default buffer settings appear to be insufficiently large to accommodate the volume of traffic on the NPS network backbone. The following commands were used to increase the kernel buffer sizes.

```
sysctl -w net.core.netdev_max_backlog=10000
sysctl -w net.core.rmem_default=16777216
sysctl -w net.core.rmem_max=33554432
sysctl -w net.ipv4.tcp_mem='194688 259584 389376'
sysctl -w net.ipv4.tcp_rmem='1048576 4194304 33554432'
sysctl -w net.ipv4.tcp_no_metrics_save=1
```

With these modified kernel parameters, we reduced the packets dropped by Tcpdump to less than 1% (70121 packets dropped out of 7822944 packets). While this did not eliminate all packet loss, it reduced it to an acceptable rate for continued testing.

A. EXPERIMENT ONE

The data collected from Experiment One showed that Suricata consumed more computational resources than Snort while monitoring the same amount of network traffic. Figures 5 and 6 graph the virtual server CPU use of both Snort and Suricata while monitoring the backbone interface. CPU use for Snort is 60–70 % for one CPU while Suricata maintains an average in the 50–60% range across all four CPUs...but Suricata uses each individual CPU at a rate less than that of Snort.

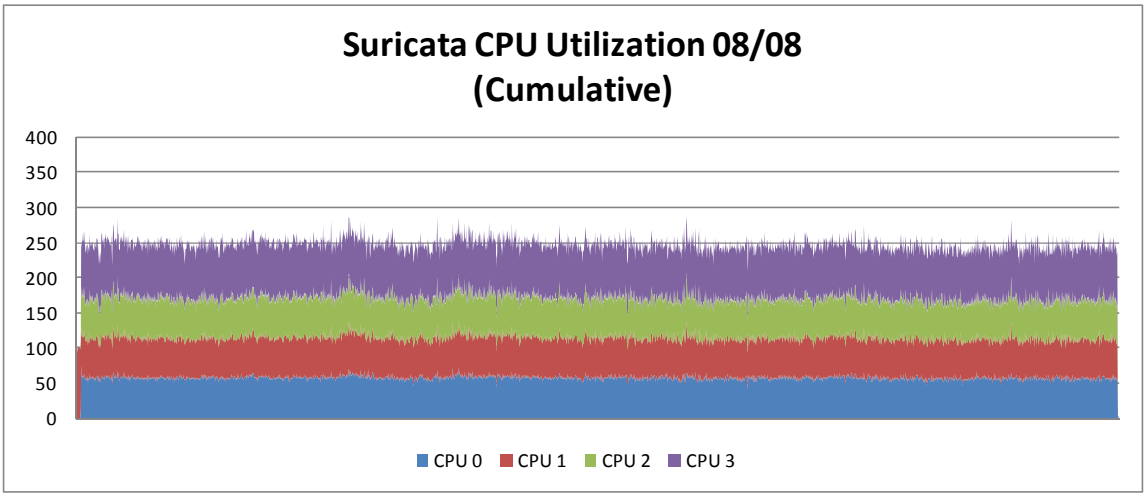

Figure 5. Suricata CPU Use

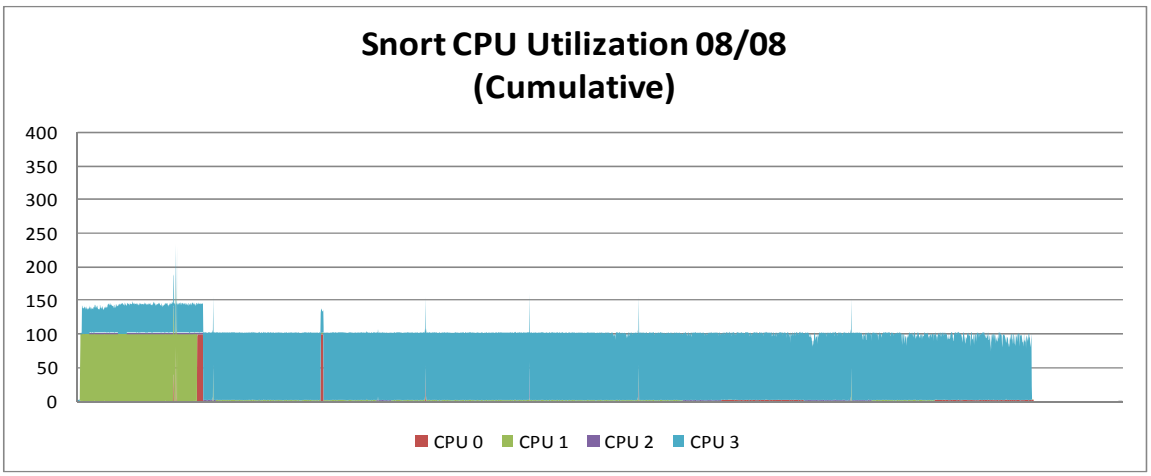

Figure 6. Snort CPU Use

Our data showed that Suricata was more memory-intensive than Snort. As illustrated in Figure 7, system memory use increased starting at approximately 1.5 Gbytes and increased to just over 3 Gbytes before tapering off near 3.3 Gbytes. Snort's memory usage was relatively low, starting at only 0.8 Gbytes and remaining below 1.0 Gbytes for the entire test period, as shown in Figure 8.

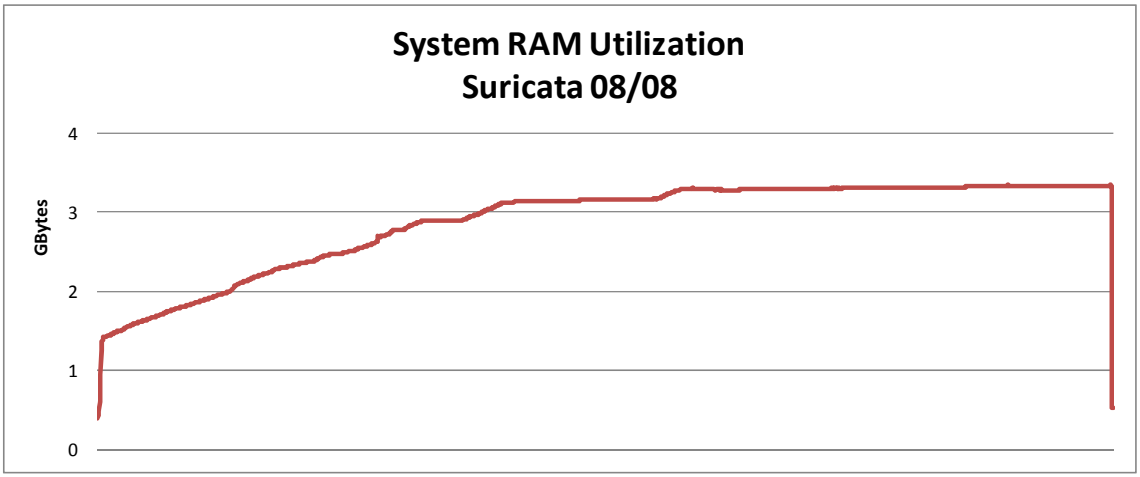

Figure 7. Suricata RAM Use

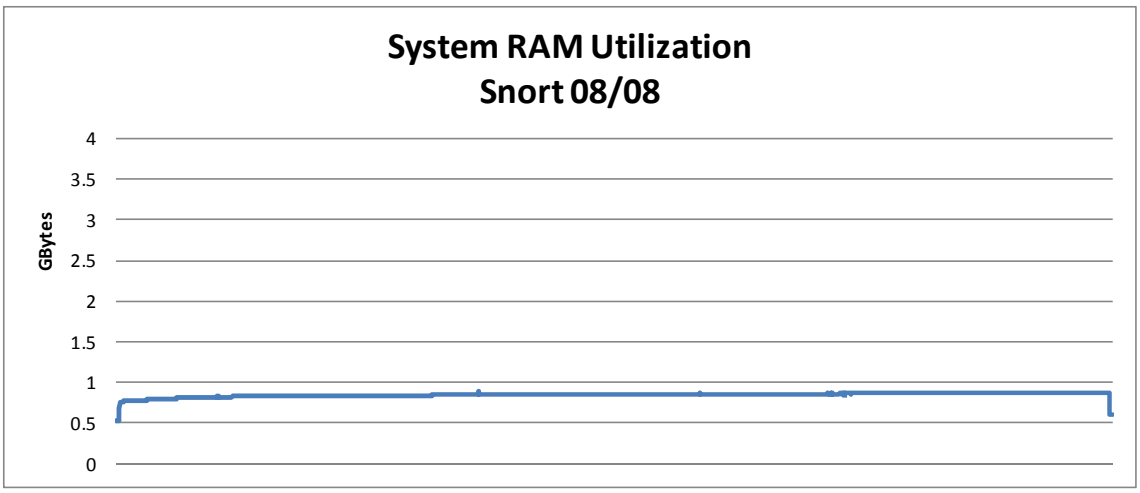

Figure 8. Snort RAM Use

Network performance remained an issue for our tests with Snort. While we were able to reduce the packet drop rate in Tcpdump to less than 1%, the output log from Snort reported a packet drop rate of 53%. Suricata, on the other hand, had a drop rate of 7%. The Snort log file further classifies the dropped packets as "outstanding packets," meaning packets that are dropped before being received by the packet processing engine in Snort (Watchinski, 2010). Our data showed that the number of outstanding packets in Snort matched the number of dropped packets in Snort, indicating that the loss of packets occurred prior to packet capture, and is therefore not a function of the processing load of the detection engine itself. Suricata does not break down the composition of dropped packets in the same manner as Snort, so the same deduction cannot be assumed solely by the log files. Further investigation and research should be conducted to determine why there is a disparity between the rate of dropped packets in Tcpdump, Suricata, and Snort.

Network performance during the first experiment was not comparable since the detection engines were not monitoring the same network traffic at the same time. The average packet rate during the Suricata period was 33,731 packets per second, and during the Snort period was 20,090 packets per second. Figures 9 and 10 illustrate the observed packet rate variation between the Snort and the Suricata.

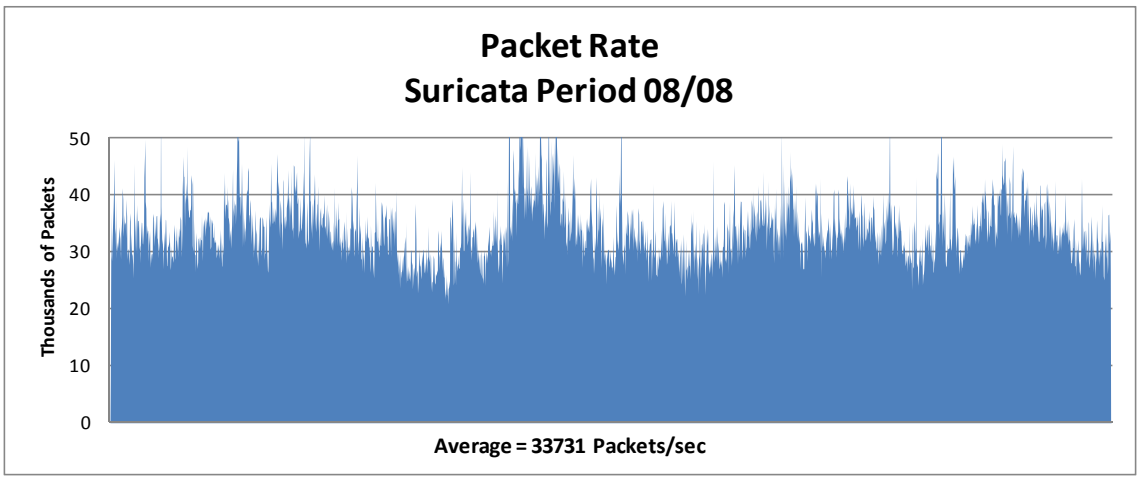

Figure 9. Suricata Packet Rate

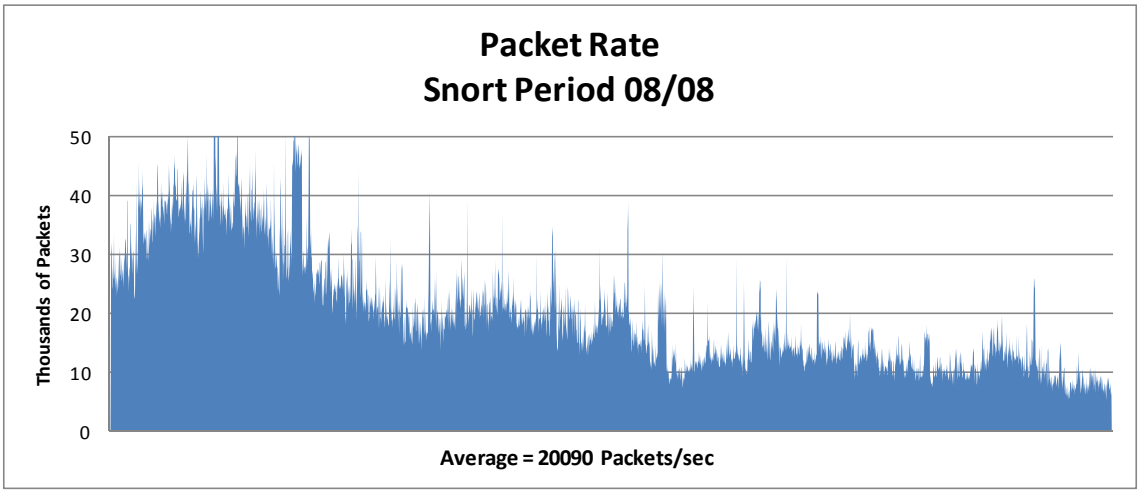

Figure 10. Snort Packet Rate

Experiment One also measured the alerts generated by Suricata and Snort running simultaneously on the same system. Figure 11 and Appendix A, Annex 1, compare the alerts generated by the two detection engines.

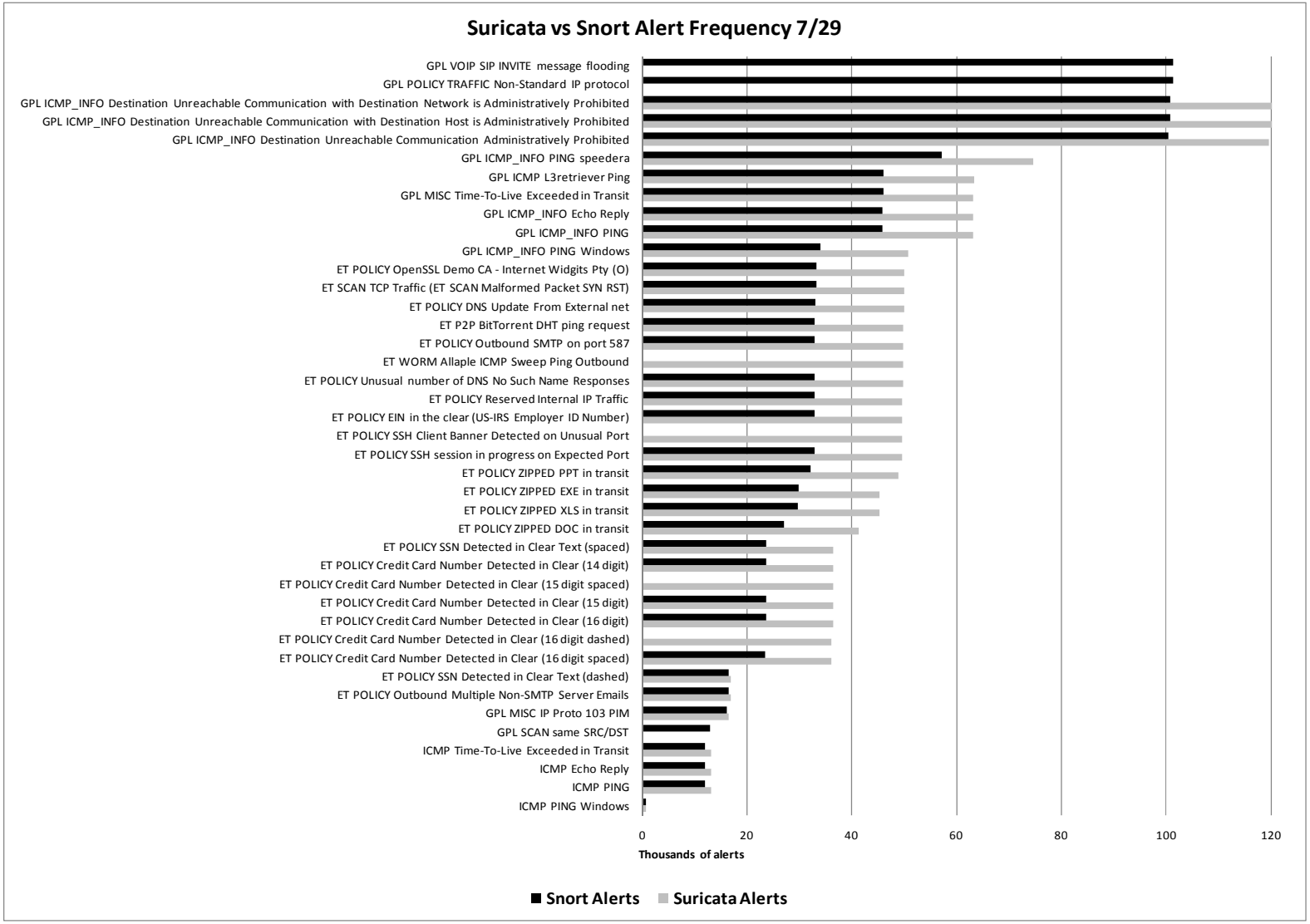

Figure 11. Suricata and Snort Combined Alert Frequency

The data show that for most rules Suricata generated more alerts than Snort on the same network traffic. Though both engines loaded the same rule sets, we did get error messages and some rules may have failed to load successfully on one engine. Other reasons could be a bug in the implementation of the rule on an engine, or a problem in the algorithm used to analyze the traffic. Further study at the packet level would be necessary to determine exactly what happened in each case.

B. EXPERIMENT TWO

Experiment Two ran Suricata on the Hamming supercomputer. We tested a number of configuration settings for the number of processing threads and the run mode. Installation of Snort and Suricata on Hamming was straightforward, with the only difference from Experiment One being the relocation of the libraries and binaries to a user-accessible directory. For these experiments, we used a 6GB Libpcap file previously generated from NPS backbone traffic. We did not study the accuracy of the alerts for this experiment, only the relative difference in processing performance.

We adjusted three parameters in the Suricata configuration file to tune the performance: detect_thread_ratio, max-pending-packets, and run mode (OISF, 2010).

- The *detect_thread_ratio* value determines the number of threads that Suricata will generate within the detection engine. Detect_thread_ratio is multiplied by the number of CPUs available to determine the number of threads. The default detect_thread_ratio in Suricata is 1.5. In our experiments we used values from .1 through 2.0.

- The *max-pending-packets* value determines the maximum number of packets the detection engine will process simultaneously. There is a tradeoff between caching and CPU performance as this number is increased. While increasing this number will more fully use multiple CPUs it will also increase the amount of caching required within the detection engine. The default for max-pending-packets is 50. In our experiments we increased this value by an order of 10 for each iteration up to 50,000.

- The *runmode* value determines how Suricata will handle the processing of each thread. There are three options: single, auto, and autofp. Single instructs Suricata to operate in single-threaded mode. In auto mode Suricata takes packets from a single flow and distributes them among the various detect threads. In autofp mode all packets from a single flow are assigned to a single detect thread.

Results from our experiment showed that with 48 CPUs the difference between the performance in the auto and autofp *runmode* increased as we increased the *max-pending-packets* variable across all *detect_thread_ratio* settings. We found that the *detect_thread_ratio* setting had minimal impact on performance in either the auto or autofp *runmode* regardless of the *max-pending-packets* setting. Figure 12 illustrates the performance difference between the auto and autofp *runmode* averaged across all of the *detect_thread_ratio* settings as measured in thousands of packets per second.

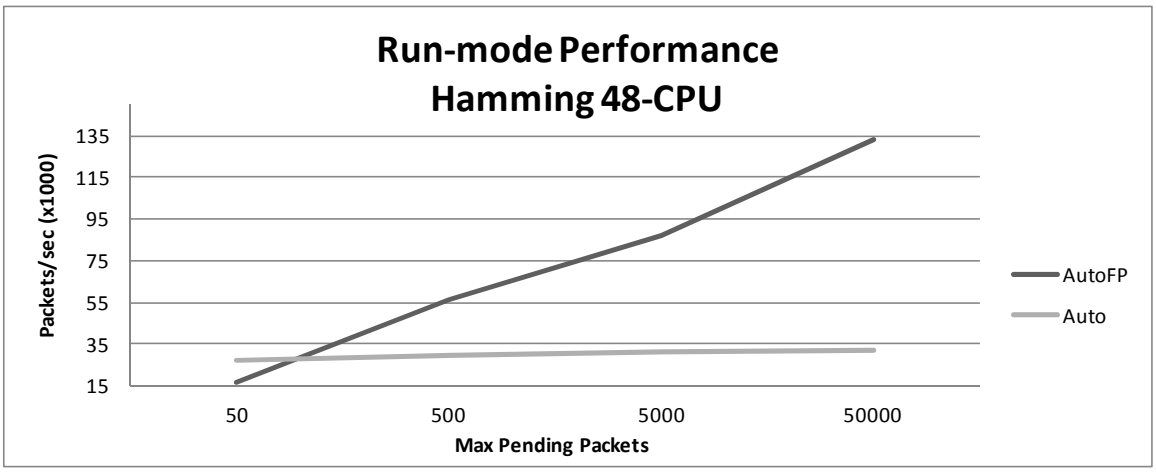

Figure 12. Suricata runmode performance for 48 CPUs

On our 4-CPU virtual machine testbed running Suricata we did not see the same performance increase observed on the 48 CPU Hamming computer when adjusting the *max-pending-packets*. As Figure 13 illustrates, our observations showed that running in AutoFP *runmode* on a 4 CPU machine incurs a performance penalty over the Auto *runmode*. Performance for both AutoFP and Auto *runmodes* averaged around 19,000 packets per second.

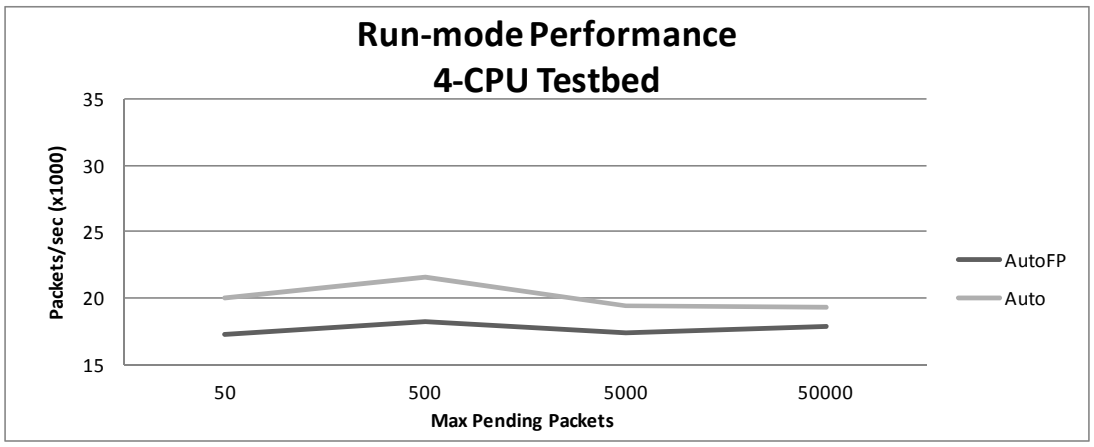

Figure 13. Suricata runmode performance for 4 CPUs

Our 6GB Libpcap file consisted of nearly 8 million packets with an average size of 803.6 bytes each. Based on these average statistics, the minimum packets per second processed by Suricata in Figure 13 equates to 108 Mbps, and the maximum packets per second on the Figure corresponds to 854 Mbps.

Unlike operations on the 48-CPU Hamming, we did observe an improvement in performance of the Auto *runmode* as we increased the *detect_thread_ratio* resulting in an increase in the number of threads. In Figure 14 we see that the noticeable drop in performance at 8 threads while in the AutoFP *runmode* is the result of limited system memory causing the operating system to begin using the hard drive swap space to augment the RAM.

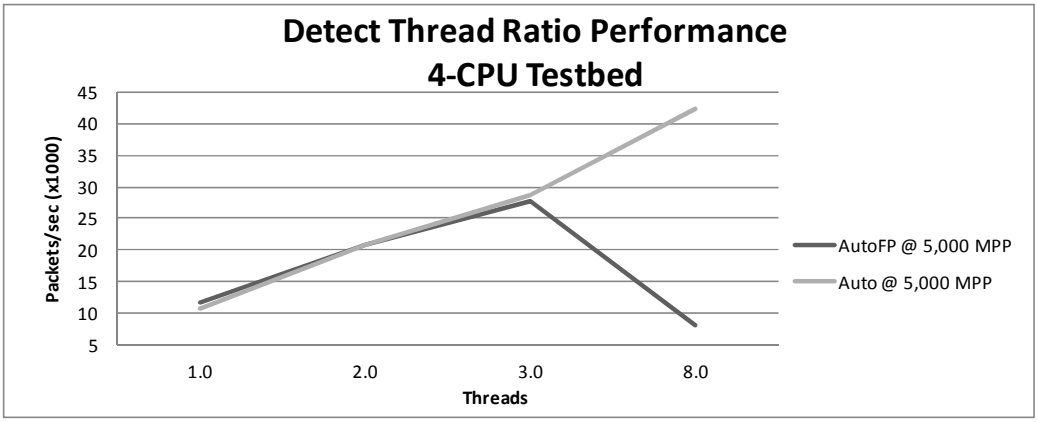

Figure 14. Suricata detect thread ratio performance 4-CPU

C. EXPERIMENT THREE

Experiment Three evaluated the accuracy of Snort and Suricata when exposed to known malicious packets. For 54 tests in 9 categories conducted against both Suricata and Snort, Suricata had 12 and Snort 16 false negatives where they did not detect the malicious traffic. Where false negatives were observed in both Suricata and Snort, the most likely reason was that the rule used by both was not loaded or optimized for the particular threat. In only two cases did Snort and Suricata report different results: One was a client-side attack where Suricata detected all 5 tests and Snort only 3, and the other was in the evasion-technique attack where Snort identified that an evasion attempt was underway; while Suricata did not.

False positives were more difficult to measure considering the composite nature of the Pytbull tests. For example, Test 8 under the category Test Rules is a full SYN scan. A true positive result would be an alert that a full SYN scan was underway. However a full SYN scan will itself generate a large number of more specific alerts that are also helpful warnings. To address this we have created a category called "Grey Positive" for an alert that could be perceived as either a false positive or a true positive depending on the context. If an alert clearly is a false positive, such as an alert for a Trojan infection during an Nmap SYN scan, then we will categorize it as a false positive. However, if an alert is generated for an attempted scan of the VNC protocol while conducting the Nmap SYN scan, the alert is an indicator of a scan, and could therefore be considered a "grey positive."

Tables 1 and 2 summarize our results. Snort generated 10 false positives and Suricata 8; the difference was in the fragmented packets category where Snort had 2 false positives and Suricata had none. This low number of false positives can be accounted for when looking at our "grey positive" category. Snort had a total of 1168 grey positives and Suricata 1449. The majority of these came from the category of evasion techniques because the tests consisted of five separate port scans, so the amount of traffic generated for this test alone was significantly more than any other test.

Refer to Appendix A Annex 2 for a sample of the detailed results of each test.

Test Category	Snort False -	Snort False +	Snort Grey +	Snort True +	Snort Other
Client Side Attacks	5	0	0	0	10
Test Rules	0	4	61	11	7
Bad Traffic	2	0	5	0	3
Fragmented Packets	3	2	0	0	4
Multiple Failed Logins	0	0	0	3	3
Evasion Techniques	1	3	1640	13	20
Shell Codes	4	1	9	16	13
Denial of Service	0	0	5	1	1
PCAP Replay	1	0	0	0	1
Total	**16**	**10**	**1168**	**44**	**62**

Table 1. Summary of Snort Alerts

Test Category	Suricata False -	Suricata False +	Suricata Grey +	Suricata True +	Suricata Other
Client Side Attacks	0	0	9	16	16
Test Rules	0	4	68	12	12
Bad Traffic	2	0	5	0	6
Fragmented Packets	3	0	4	2	9
Multiple Failed Logins	0	0	4	0	2
Evasion Techniques	2	3	1275	12	29
Shell Codes	4	1	4	38	25
Denial of Service	0	0	5	1	2
PCAP Replay	1	0	0	0	2
Total	**12**	**8**	**1449**	**81**	**103**

Table 2. Summary of Suricata Alerts

In calculating the recall and precision for our experiment, we calculated both a pure precision, consisting of only false and true positive, and a realistic precision, which combined the false and grey positives. Table 3 shows the recall and precision for our tests.

Application	Recall	Precision
Snort	.73	.81
Snort (grey and false +)		.036
Suricata	.87	.91
Suricata (grey and false +)		.052

Table 3. Snort and Suricata Recall and Precision

We now give an analysis of the detection success in each category of attack.

- Client-Side Attacks – These tests simulated the actions of a user downloading an infected file from the Internet. We conducted 5 Client Side Attack tests where we downloaded 4 infected PDF files and 1 infected XLS file across our network and through the intrusion-detection system. In all 5 cases Suricata generated true-positive alerts from the malicious downloads. On the same rule set Snort did not generate any alerts. There were a number of additional alerts that were false positives generated during the test including alerts for a file transfer (a necessary function of the Pytbull application to record the alert data), a tilde character in the URI, and a successful FTP login (a necessary function used by Pytbull to retrieve the alert data).

- Test Rules – This test evaluated how well the detection engine responded to a variety of different probes into a network. Included are Local File Inclusion (LFI) attacks, various network scans, SQL injections, and reverse shell attempts. For these tests Pytbull used HTTP requests, along with the Nmap, Netcat, and Nikto applications. Both Suricata and Snort generated a number of true positive and false positive alerts for the test traffic. In test number 9 Pytbull generates an Nmap full-connect scan across all 65535 ports. While this was correctly identified and alerted by both Suricata and Snort as an Nmap scan, both detection engines also generated an alert for a potential VNC scan of ports 5800-5820, which is understandable since Nmap is scanning VNC ports too. During the same

scan both Suricata and Snort generated a false positive alert for a possible network Trojan attack, which was in fact not occurring. While these false positive alerts reflected a valid event (scanning of VNC ports), they could be a distraction from the larger overall picture that the entire network was being scanned.

- Bad Traffic – These tests consisted of malformed network packets in which either the flags in the TCP header were not set correctly or the type of packet did not match its header. This test used Nmap and Scapy to generate malformed packets. Both Suricata and Snort only alerted on 1 out of the 3 malformed packet tests, and neither generated an alarm on the Scapy-modified packets. The first packet was a common "Xmas scan" generated by Nmap; the second used Scapy to modify the IP protocol flag to indicate version 3; and the third changed the source and destination port to the same number.

- Fragmented Packets – For these tests Pytbull implemented two types of fragmented packet attacks, a Ping-of-Death attack where the packet fragments when reassembled are larger than allowed in the protocol specification, and a Nestea attack where the order of reassembly is out of sequence. Both Suricata and Snort were unable to detect the Nestea attack, and Snort generated a false-positive alert for an outbound SSH scan. Suricata alone detected the Ping-of-Death attack.

- Multiple Failed Logins Using a known bad username and password combination, Pytbull attempted to log into the server multiple times. Suricata generated a false positive alert for each of these attempts as a regular login attempt but not as a failed attempt. Snort generated an "FTP Bad Login" alert for each one.

- Evasion Techniques – These tests employed common techniques for evading detection engines. The first two used the decoy function within Nmap to obscure the source address of the attacker by hiding it within a

number of other IP addresses. We were unable to obtain accurate results from this test because our installation of Pytbull was on a virtual machine that performed network-address translation, so the attempts by Nmap to use different IP addresses resulted in the same IP address, defeating the evasion attempt. The next test used hexadecimal encoding to attempt to evade the detection engines; neither Snort nor Suricata detected it. Test 28 used Nmap to generate small fragments for the TCP portion of the packets in an attempt to overwhelm the detection engine with reassembly tasks. Both detection engines were able to detect that a scan was occurring, however, Snort was the only one that identified the use of Nmap scripting and generated an appropriate alert. In tests 29 through 38 Pytbull used the various evasion techniques within the Nikto web application scanner to attempt an evasion of the detection engine. Both Suricata and Snort were effective in detecting the scans; however in only two of the tests (30 and 36) could they identify the scan as Nikto-specific. On all of the other tests both Suricata and Snort alerted on the web scanning activity but did not identify the scans as Nikto. Finally, in test 39 Pytbull used JavaScript obfuscation to attempt evasion of the detection engines. Snort alerted but Suricata did not.

- Shell Codes – Pytbull sent 13 different shellcode attacks through the detection engine. Of the 13 attempts, Suricata detected 10 and Snort detected 9. The three shell code attempts missed by Suricata were also missed by Snort, and were 1)" IRIX SGI + NOOP," 2) Buffer Overflow Attempt, and 3) Cisco VTY creation, password creation, and privilege escalation. In addition to these three, Snort also missed the "x86 setgid 0 && setuid 0."

- Denial of Service – Denial-of-service attacks are difficult to test without causing a true denial of service. Pytbull has two that it can perform. One uses the utility hping to generate an ICMP ping flood to the target machine, and one is an attempt to attack a specific application (MSSQL in

this case) with a DoS attack. We were unable to perform the hping DoS attack as that would have caused an actual denial of service on our network. Both Suricata and Snort detected the MSSQL DoS attempt, however neither one identified it as a DoS-specific attack. Instead, both alerted on suspicious traffic sent to the MSSQL TCP port 1433.

- Pcap Replay – This is the replay of a previously captured malicious payload to test how well the intrusion-detection system engine can detect other malware. For our test we used only one Pcap capture containing a sample of the Slammer worm. In this test neither Snort nor Suricata detected the Slammer worm code.

To summarize Experiment 3, when Suricata and Snort were loaded using the same rule set, in some cases both failed to generate alerts on known malicious traffic. When both failed, we can be fairly confident that this can be attributed to the rules and not the detection engines. In a few cases there were discrepancies between the Snort and Suricata alerts. One explanation could be differences in the implementation of the rule language between Snort and Suricata. Presently Suricata version 1.1 beta 2 does not support the "file_data" rule keyword, and rules in the VRT rule set that use it cause an error when loaded. Another explanation could be in the implementation of the detection algorithm within each application which could affect how the detection engine examines packets, but it is hard to obtain details of the implementations.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSION

A. DISCUSSION

We evaluated two open-source network-based intrusion-detection systems for the NPS environment. Snort is currently the de-facto standard for open-source network-based intrusion-detection systems around the world (SourceFire, 2011). Suricata is still in early stages of development but offers speed improvements and capabilities unavailable in Snort.

Both Suricata and Snort are very capable intrusion-detection systems, each with strengths and weaknesses. We tested Suricata and Snort on similar data to provide an informed recommendation to the Information Technology Department of the Naval Postgraduate School on whether to use Suricata as an additional layer of defense for the Educational Research Network. Both Suricata and Snort performed well during tests. Both did have false positives and false negatives, but much of that can be attributed to weaknesses of the rule set used for the tests. It was inconclusive from our tests whether Suricata or Snort has a better detection algorithm.

Suricata's multi-threaded architecture requires more memory and CPU resources than Snort. We saw that the aggregate CPU use of Suricata was nearly double that of Snort, and Suricata used over double the amount of RAM used by Snort. This could be attributed to the overhead required to manage the multiple detection threads in Suricata. Suricata has the advantage that it can grow to accommodate increased network traffic without requiring multiple instances. Snort is lightweight and fast but limited in its ability to scale beyond 200-300 Mbps network bandwidth per instance. While Snort's processing overhead is less than that of Suricata, the need for multiple instances to accomplish what Suricata can achieve with its multi-threaded design elevates the cost to operate and manage a Snort environment.

Experiment Two showed a big improvement in the performance of Suricata on 48 CPUs, but only by increasing the configuration variable *max-pending-packets* while in the *autofp* run-mode.

Experiment Three reinforced the importance of a well-tuned rule set for a system. In our test, both detection engines missed several common malicious payloads that should have been detected. Had the rules been properly tuned for the environment the false negative rate would have been less with a corresponding increase in true positive alerts.

Operating an intrusion-detection system on a virtual host introduced additional complications. In our experiments, we had problems with network throughput when monitoring the live network traffic from the 20Gbps network backbone. Further investigation into the network hardware used with the ESXi server is required to diagnose the cause for the high number of dropped packets on Suricata, Snort and Tcpdump. However, operational deployment of intrusion detection in a virtual host is unnecessary at NPS so these issues may be moot.

During our research the Suricata development team released three minor version changes (1.0.3, 1.0.4 and 1.0.5) and two beta versions (1.1 beta1 and 2) of the next minor version change. Each version contained significant improvements to the previous version, illustrating the rapid advancement of the detection engine. By comparison, Snort has been on the same production release (2.9.0.5) for 5 months. Rapid development requiring frequent upgrades is not an optimal choice for a production environment intrusion detection, so that is a weakness of Suricata. Nonetheless, the pace of upgrades is likely to slow and Suricata should be more reliable.

B. RECOMMENDATION

Suricata is a very capable intrusion-detection system and should be used to augment the existing Snort system at the Naval Postgraduate School. The ability to use multi-threaded techniques in a multiple-CPU environment will give Suricata an advantage over single-threaded detection engines like Snort as the network throughput at NPS continues to increase.

Snort is still very capable and should remain in use within the NPS production environment for the immediate future. But as the actual bandwidth on the NPS ERN backbone continues to grow to rates greater than 200 Mbps, the single-threaded Snort

architecture will not be able to keep up with the network load (Lococo, 2011). Deploying both Snort and Suricata today will mean an easier transition to the multi-thread design of Suricata as the network load begins to overwhelm the existing Snort infrastructure.

Network intrusion-detection systems are just one security technology, and we must also incorporate host-based systems so that we can catch the percentage of threats that are missed by firewalls and other network-monitoring systems. A weakness of both is the reliance on signatures for detection. While signatures will detect most of the known malicious traffic in an enterprise, they cannot detect something that has not been seen before. For this we must additionally use anomaly-based intrusion-detection systems. To support the distributed deployment of intrusion-detection systems Suricata should consider incorporating SNMP traps as an additional means to deliver alerts to the event management console.

C. FUTURE RESEARCH

There are a number of areas for future research involving intrusion detection. As attempts to compromise our information become more and more complex, it will become more difficult to detect these new threats. As we increase the number of sensors distributed throughout our networks, the task of managing and correlating the information produced by these sensors grows. If we are to be effective at monitoring our networks and guarding the information that resides therein, we must make a concerted effort to become aware of everything on the network. Alert and event correlation is a step in that direction, and improving how we monitor and interpret that information is worthy of future research.

Within the intrusion-detection category specifically, additional research should be performed in the incorporation of signature and anomaly-based intrusion detection to meet the unknown and persistent threats to our information and data infrastructure. Presently the two technologies are usually separate and require independent implementations. Future research should address the integration of both signature and anomaly-based intrusion detection into a unified and seamless solution.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A

ANNEX 1 TABLE OF COMBINED SURICATA AND SNORT ALERTS DURING TESTING

The following table is a summary of the alerts generated during the variation of experiment 1: the combined run of Suricata and Snort while observing the same ERN network traffic. The SID is the Snort ID number of the rule associated with the description in the next column. The Difference column is the difference between the number of alerts generated by Suricata and Snort, and is represented as a percentage in the last column. This data was collected over a 4 hour period where both detection engines were started and stopped within one second of each other.

Comparison of Snort and Suricata Alerts 07/27					
SID	Alert Description	Suricata Alerts	Snort Alerts	Difference	%
382	ICMP PING Windows	805	791	14	2%
384	ICMP PING	13112	11986	1126	9%
408	ICMP Echo Reply	13115	11989	1126	9%
449	ICMP Time-To-Live Exceeded in Transit	13122	11996	1126	9%
527	GPL SCAN same SRC/DST		12940		
2189	GPL MISC IP Proto 103 PIM	16481	16229	252	2%
2000328	ET POLICY Outbound Multiple Non-SMTP Server Emails	16842	16535	307	2%
2001328	ET POLICY SSN Detected in Clear Text (dashed)	16844	16536	308	2%
2001375	ET POLICY Credit Card Number Detected in Clear (16 digit spaced)	36100	23396	12704	35%
2001376	ET POLICY Credit Card Number Detected in Clear (16 digit dashed)	36101			
2001377	ET POLICY Credit Card Number Detected in Clear (16 digit)	36400	23571	12829	35%
2001378	ET POLICY Credit Card Number Detected in Clear (15 digit)	36411	23578	12833	35%
2001379	ET POLICY Credit Card Number Detected in Clear (15 digit spaced)	36414			
2001381	ET POLICY Credit Card Number Detected in Clear (14 digit)	36414	23579	12835	35%
2001384	ET POLICY SSN Detected in Clear Text (spaced)	36429	23581	12848	35%
2001402	ET POLICY ZIPPED DOC in transit	41323	27116	14207	34%
2001403	ET POLICY ZIPPED XLS in transit	45286	29724	15562	34%
2001404	ET POLICY ZIPPED EXE in transit	45369	29800	15569	34%
2001405	ET POLICY ZIPPED PPT in transit	48915	32161	16754	34%
2001978	ET POLICY SSH session in progress on Expected Port	49645	32805	16840	34%
2001980	ET POLICY SSH Client Banner Detected on Unusual Port	49646			
2002658	ET POLICY EIN in the clear (US-IRS Employer ID Number)	49655	32813	16842	34%
2002752	ET POLICY Reserved Internal IP Traffic	49656	32814	16842	34%
2003195	ET POLICY Unusual number of DNS No Such Name Responses	49726	32879	16847	34%
2003292	ET WORM Allaple ICMP Sweep Ping Outbound	49731			
2003864	ET POLICY Outbound SMTP on port 587	49733	32881	16852	34%
2008581	ET P2P BitTorrent DHT ping request	49734	32882	16852	34%
2009702	ET POLICY DNS Update From External net	49981	33120	16861	34%
2011368	ET SCAN TCP Traffic (ET SCAN Malformed Packet SYN RST)	50030	33164	16866	34%
2011540	ET POLICY OpenSSL Demo CA - Internet Widgits Pty (O)	50031	33166	16865	34%
2100382	GPL ICMP_INFO PING Windows	50836	33957	16879	33%
2100384	GPL ICMP_INFO PING	63141	45942	17199	27%
2100408	GPL ICMP_INFO Echo Reply	63144	45944	17200	27%
2100449	GPL MISC Time-To-Live Exceeded in Transit	63151	45950	17201	27%
2100466	GPL ICMP L3retriever Ping	63261	46060	17201	27%
2100480	GPL ICMP_INFO PING speedera	74651	57143	17508	23%
2100485	GPL ICMP_INFO Destination Unreachable Communication Administratively Prohibited	119536	100348	19188	16%
2100486	GPL ICMP_INFO Destination Unreachable Communication with Destination Host is Administratively Prohibited	120045	100832	19213	16%
2100487	GPL ICMP_INFO Destination Unreachable Communication with Destination Network is Administratively Prohibited	120045	100832	19213	16%
2101620	GPL POLICY TRAFFIC Non-Standard IP protocol		101343		
100000158	GPL VOIP SIP INVITE message flooding		101344		

ANNEX 2 EXAMPLE PYTBULL REPORTS FOR SURICATA AND SNORT

Category: Client Side Attacks

Test 4: Corrupt PDF File (CVE2009-4324)

Snort False Negative: Both alerts generated by Snort in this example are due to the Pytbull process of obtaining the alert data from the intrusion-detection system. Neither alert relates to the exploited PDF file that was transmitted in Test 4.

Test num	4
Time	2011-08-10 16:38:55.558577
File	004e74d54dcf79c641d5cf8a615488a0
Alerts	[**] [1:2002822:8] ET POLICY Wget User Agent [**] [Classification: Attempted Information Leak] [Priority: 2] 08/10-16:38:55.936680 172.20.21.180:49694 -> 172.20.105.57:80 TCP TTL:64 TOS:0x0 ID:8121 IpLen:20 DgmLen:210 DF ***AP*** Seq: 0xAD114A01 Ack: 0x6267CE85 Win: 0x6 TcpLen: 32 TCP Options (3) => NOP NOP TS: 191124919 2289961616 [Xref => http://www.emergingthreats.net/cgi-bin/cvsweb.cgi/sigs/POLICY/POLICY_Web_Crawling][Xref => http://doc.emergingthreats.net/2002822][Xref => http://www.gnu.org/software/wget] [**] [1:2001117:5] ET DNS Standard query response, Name Error [**] [Classification: Not Suspicious Traffic] [Priority: 3] 08/10-16:38:57.952344 172.20.20.11:53 -> 172.20.21.180:34090

Suricata True Positive: Suricata generated an alarm based on the PDF file containing JavaScript.

Test num	4
Time	2011-08-10 13:27:02.833020
File	004e74d54dcf79c641d5cf8a615488a0
Alerts	08/10/2011-13:27:03.506887 [**] [1:2010736:2] ET FTP FTP RETR command attempt without login [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 172.20.114.183:55487 -> 172.20.21.180:21 08/10/2011-13:27:03.516970 [**] [1:2002822:9] ET POLICY Wget User Agent [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 172.20.21.180:55215 -> 172.20.105.57:80 08/10/2011-13:27:03.516948 [**] [1:2009954:8] ET WEB_SERVER Tilde in URI after file, potential source disclosure vulnerability [**] [Classification: Web Application Attack] [Priority: 1] {TCP} 172.20.105.57:80 -> 172.20.21.180:55215 08/10/2011-13:27:03.517175 [**] [1:2010882:7] ET POLICY PDF File Containing Javascript [**] [Classification: Misc activity] [Priority: 3] {TCP} 172.20.105.57:80 -> 172.20.21.180:55215 08/10/2011-13:27:03.517175 [**] [1:18681:2] POLICY download of a PDF with embedded JavaScript - JavaScript string [**] [Classification: Potential Corporate Privacy Violation] [Priority: 1]

Category: Test Rules

Test #6: Simple LFI Attack

Snort True Positive: Snort generated an alert based on the '/etc/passwd' string passed through an HTTP command.

Test num	6
Time	2011-08-10 16:39:01.654016
Test name	Simple LFI
Port	80/tcp
Payload	GET /index.php?page=../../../etc/passwd HTTP/1.1 Host: 127.0.0.1 User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.5) Gecko/20041202 Firefox/1.0
Sig matching	OK Used pattern: 1:1122:
Alerts	[**] [1:1122:6] GPL ATTACK_RESPONSE /etc/passwd [**] [Classification: Attempted Information Leak] [Priority: 2] 08/10-16:39:02.030511 172.20.114.183:51849 -> 172.20.21.180:80 TCP TTL:63 TOS:0x0 ID:56369 IpLen:20 DgmLen:219 DF ***AP*** Seq: 0xAFE3B75B Ack: 0xAE3007AA Win: 0xE5 TcpLen: 32 TCP Options (3) => NOP NOP TS: 7367541 191131013 [**] [1:2002567:4] ET POLICY HTTP - Password [**] [Classification: Potential Corporate Privacy Violation] [Priority: 1] 08/10-16:39:02.030511 172.20.114.183:51849 -> 172.20.21.180:80 TCP TTL:63 TOS:0x0 ID:56369 IpLen:20 DgmLen:219 DF ***AP*** Seq: 0xAFE3B75B Ack: 0xAE3007AA Win: 0xE5 TcpLen: 32

Suricata True Positive: Suricata generated an alert based on the '/etc/passwd' string passed through an HTTP command.

Test num	6
Time	2011-08-10 13:27:08.933395
Test name	Simple LFI
Port	80/tcp
Payload	GET /index.php?page=../../../etc/passwd HTTP/1.1 Host: 127.0.0.1 User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.5) Gecko/20041202 Firefox/1.0
Sig matching	OK Used pattern: 1:1122:
Alerts	08/10/2011-13:27:09.616437 [**] [1:1122:9] WEB-MISC /etc/passwd [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 172.20.114.183:51429 -> 172.20.21.180:80 08/10/2011-13:27:09.615915 [**] [1:1122:9] WEB-MISC /etc/passwd [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 172.20.114.183:51429 -> 172.20.21.180:80 08/10/2011-13:27:09.615915 [**] [1:2002567:4] ET POLICY HTTP - Password [**] [Classification: Potential Corporate Privacy Violation] [Priority: 1] {TCP} 172.20.114.183:51429 -> 172.20.21.180:80 08/10/2011-13:27:11.632235 [**] [1:2003303:3] ET POLICY FTP Login Attempt (non-anonymous) [**] [Classification: Misc activity] [Priority: 3] {TCP} 172.20.114.183:52761 -> 172.20.21.180:21 08/10/2011-13:27:11.641406 [**] [1:2010736:2] ET FTP FTP RETR command attempt without login [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 172.20.114.183:52761 -> 172.20.21.180:21

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B

ANNEX 1 EMERGING THREATS (ET) AND VULNERABILITY RESEARCH TEAM (VRT) RULE CATEGORIES USED IN EXPERIMENTS.

- emerging-all.rules	- policy.rules
- attack-responses.rules	- pop2.rules
- backdoor.rules	- pop3.rules
- bad-traffic.rules	- rpc.rules
- blacklist.rules	- rservices.rules
- botnet-cnc.rules	- scada.rules
- chat.rules	- scan.rules
- content-replace.rules	- shellcode.rules
- ddos.rules	- smtp.rules
- dns.rules	- snmp.rules
- dos.rules	- specific-threats.rules
- exploit.rules	- spyware-put.rules
- finger.rules	- sql.rules
- ftp.rules	- telnet.rules
- icmp-info.rules	- tftp.rules
- icmp.rules	- virus.rules
- imap.rules	- voip.rules
- info.rules	- web-activex.rules
- misc.rules	- web-attacks.rules
- multimedia.rules	- web-cgi.rules
- mysql.rules	- web-client.rules
- netbios.rules	- web-coldfusion.rules
- nntp.rules	- web-frontpage.rules
- oracle.rules	- web-iis.rules
- other-ids.rules	- web-misc.rules
- p2p.rules	- web-php.rules
- phishing-spam.rules	- x11.rules

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

B, T. (2010). Snort. Message posted to www.digitalboundary.net/wp/?p=90

Ben-Kiki, O., Evans, C. & dot Net, I. (2010). *YAML.* Retrieved 8/19/2011, from yaml.org

CPU limit. (2011). Retrieved 8/18/2011 from cpulimit.sourceforge.net

Damaye, S. (2011a). *Pytbull.* Retrieved 8/19/2011, from pytbull.sourceforge.net

Damaye, S. (2011b). Suricata-vs-snort. Message posted to www.aldeid.com/wiki/Suricata-vs-snort

Day, D., & Burns, B. (2011). A performance analysis of snort and suricata network intrusion detection and prevention engines. *IDCS 2011, the Fifth International Conference on Digital Society,* Gosier, Guadeloupe, France. 187–192.

Emerging Threats. (2011). *Emerging threat.* Retrieved 8/19, 2011, from www.emergingthreats.net

Fossl, M. (2011). *Symantec Internet security threat ReportTrends for 2010.*Symantec Corp.

García-Teodoro, P., Díaz-Verdejo, J., Maciá-Fernández, G., & Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security, 28*(1-2), 18-28. doi:DOI: 10.1016/j.cose.2008.08.003.

Haferman, J. (2011). *NPS high performance computing center.* Retrieved 8/19/2011, from www.nps.edu/Technology/HPC/Images/HPC_brochure0209.pdf

Jonkman, M. (2009). Suricata IDS available for download. Message posted to marc.info/?l=snort-users&m=126229065928554&w=2.

Kuipers, D., & Fabro, M. (2006). *Control system cyber security: Defense in depth strategies.* No. INL/EXT-06-11478).

Kumar, G., & Panda, S. N. (2011). Spectrum of effective security trust architecture to manage interception of packet transmission in value added networks. *The Global Journal of Computer Science and Technology, 11*(4), 7377

Leblond, E. (2011). Optimizing Linux on multicore CPUs. Message posted to home.regit.org/2011/01/optimizing-suricata-on-a-multicore-cpu

Lococo, M. (2011). Capacity planning for snort. Message posted to mikelococo.com/2011/08/snort-capacity-planning

Moore, G. (1965). Cramming more components on to integrated circuits. *Electronics, 38*(8).

Nielsen, J. (2010). *Nielsen's law of Internet bandwidth.* Retrieved 8/18/2011, from www.useit.com/alertbox/980405.html

Open Information Security Foundation (OISF). (2010). *Suricata IDS* (1.1 Beta2 ed.).

Open Information Security Foundation (OISF). (2011a). *Open information security foundation (OISF).* Retrieved 5/4/2011, from www.openinfosecfoundation.org

Open Information Security Foundation (OISF). (2011b). *Suricata installation notes* (1.1 Beta 2 ed.) Retrieved from www.openinfosecfoundation.org/doc/INSTALL.txt

Open Information Security Foundation (OISF). (2011c). *Suricata - multi threading.* Retrieved 5/6/2011, from redmine.openinfosecfoundation.org/projects/suricata/wiki/Multi_Threading

Ristic, I. (2009). *HTTP parser for intrusion detection and web application firewalls.* Retrieved 8/18/2011, from blog.ivanristic.com/2009/11/http-parser-for-intrusion-detection-and-web-application-firewalls.html

Roesch, M. (2005). *The story of snort: Past, present and future* Retrieved 8/18/2011, from www.net-security.org/article.php?id=860

Roesch, M. (2010). Single threaded data processing pipelines and the intel archietecture, or no performance for you, now go home. Message posted to vrt-blog.snort.org/2010/06/single-threaded-data-processing.html

Shimel, A. (2010). Is this town big enough for two IDS? Message posted to www.networkworld.com/community/node/67435

SourceFire. (2011). *Snort IDS.* Retrieved 8/19/2011, from www.snort.org

Tenhunen, T. (2008). *Implementing an intrusion detection system in the MYSEA architecture.* (Master of Computer Science, Naval Postgraduate School).

Watchinski, M. (2010). Unusual snort performance stats. Message posted to comments.gmane.org/gmane.comp.security.ids.snort.general/30527

Weber, T. (2001). *Network intrusion detection - keeping up with increasing information volume.* SANS Institute.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Neil Rowe
   Naval Postgraduate School
   Monterey, California

4. Rex Buddenberg
   Naval Postgraduate School
   Monterey, California

5. D. C. Boger
   Naval Postgraduate School
   Monterey, California